

User Manual

Colm Ó hÉigearthaigh
CASE4 - 99387212
coheig-case4@computing.dcu.ie

May 6, 2003

Abstract

This document describes the installation of the various pieces of code written in this project. It also describes how to start and use the programs. The first chapter details how to install the bloch sphere patch to the QCL, and how to use it in the QCL. The second chapter describes how to install and run the GUI on both the server and client-side. The third chapter gives instructions on how to install and use the parallel routines written for the QCL. The fourth chapter is a tutor on the Quantum Computing Language, which needs to be mastered to use both the bloch sphere and parallel code.

Contents

1	The Bloch Sphere	4
1.1	Applying the patch	4
1.2	Using the bloch sphere code inside the QCL	4
2	The Graphical User Application	7
2.1	Compiling and running the Server Code	7
2.2	Compiling and running the Client Code	8
2.3	Using the X-Server	9
2.4	The cluster information panel	11
3	Using the parallel routines of the Quantum Computing Language	14
3.1	Installing the MPI routines in QCL	14
3.2	Installing the MPI patchfile to the QCL	14
3.3	Using the MPI routines in the QCL	15
4	A tutor on the Quantum Computing Language	15
4.1	Classical components of the QCL	16
4.2	Quantum components of the QCL	17

1 The Bloch Sphere

1.1 Applying the patch

The patch for the bloch sphere against the most up-to-date version of the QCL(QCL-0.5.0) is available here(http://www.redbrick.dcu.ie/hego/project/bloch_sphere.patch). To apply this patch, copy the patch file into /tmp. Download a copy of the QCL source code and untar and unzip it with the following command;

```
tar zxvf qcl-0.5.0.tgz
```

Change directory into the qcl-0.5.0 directory. Then apply the patch, with this command;

```
patch </tmp/bloch_sphere.patch
```

The code now needs to be compiled. Firstly, alter the Makefile to suit your system. Then create the .depend file with;

```
make dep
```

To compile the code then type;

```
make
```

1.2 Using the bloch sphere code inside the QCL

The Bloch Sphere code allows you to view the state of a single qubit inside the QCL. A windowing system such as XFree86 must be in use to allow the viewing of the bloch sphere, this code cannot be run from a console, unless the graphical output is being redirected to an x-server somewhere.

First of all, start up the Quantum Computing language. To use the test program detailed below, initialise the QCL like this;

```
./qcl -i lib/bloch.qcl
```

A test program to show the usage of the bloch sphere code, is provided in the "bloch.qcl" file in the *lib* subdirectory.

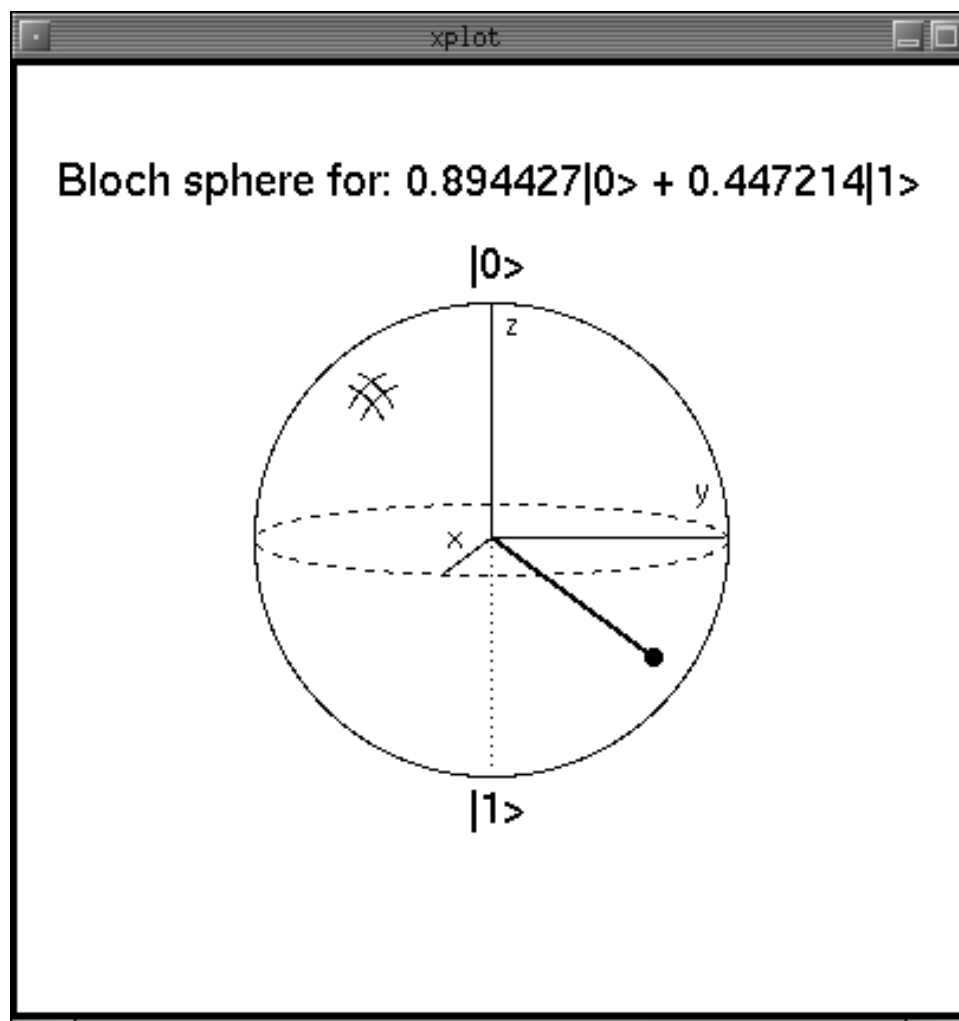
Listing 1: bloch.qcl

```
procedure bloch() {  
    qureg a [1];      // Initialise the qubit  
    Mix(a);           // Superpose the qubit  
    Rot(0.6435, a);    // Rotate the qubit so that it is |0> with probability  
                      // 0.8, and |1> with probability 0.2  
    bloch a;          // Display the qubit on the bloch sphere  
}
```

This procedure initialises a quantum register of size 1 qubit. The qubit is superposed with the *Mix* operation, ie. the Hadamard operation, with the qubit being in the resultant state of;

$$|\phi\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle \quad (1)$$

The *Rot* operation is then applied to the qubit, so that when measured, the qubit is in state $|0\rangle$ with probability 0.8 and state $|1\rangle$ with probability 0.2. The bloch sphere of that qubit is then produced with the command *bloch a*. This produces the following screenshot using XFree86 4.3;



2 The Graphical User Application

2.1 Compiling and running the Server Code

The server code is available at this location (http://www.redbrick.dcu.ie/hego/project/gui_server1.tar.bz2). Download this code into a destination directory. Unzip and untar the file with the following command;

```
bzip2 -d gui_server1.tar.bz2
tar xvf gui_server1.tar
```

The java code is stored in the directory *com/jcraft/weirdx*. To compile this code type;

```
javac com/jcraft/weirdx/*.java
```

An RMI stub file then needs to be generated for the Spy class. This can be generated with the following command;

```
rmic com.jcraft.weirdx.Spy
```

The logins.c file then needs to be compiled, and the source moved to the /tmp subdirectory. Assuming the gcc compiler is installed on the system, this can be compiled by;

```
gcc logins.c -o logins
mv logins /tmp
```

The java policy file then needs to be altered. A sample java.policy file is included with the server code. This needs to be modified to reflect the source directory;

Listing 2: java.policy

```
grant {
    permission java.net.SocketPermission "*:1024-65535", "connect,accept";
    permission java.net.SocketPermission "*:80", "connect";
};
grant codeBase "file :/home/users/hego/qcl/gui" {
    permission java.io.FilePermission "<<ALL FILES>>", "read,execute";
};
```

Before the server can be run, the rmiregistry must be started on port 8000. This is done with the following command;

```
rmiregistry 8000 &
```

Finally, a sample server file that starts the server code is included called "server4.sh". This should again be modified to reflect the system.

Listing 3: server4.sh

```
#!/bin/sh

/usr/local/j2sdk1.4.1_01/bin/java -Djava.rmi.server.hostname=136.206.19.200
-Djava.security.policy=java.policy com.jcraft.weirdx.Monitor 8000
```

2.2 Compiling and running the Client Code

First of all, weirdx must be downloaded and compiled. This is available from the following location (<http://prdownloads.sourceforge.net/weirdx/weirdx-1.0.31.tar.gz>). Unzip and untar and compile this file in a source directory with the following commands;

```
tar zxvf weirdx-1.0.31.tar.gz
cd weirdx-1.0.31
javac -O com/jcraft/weirdx/*.java com/jcraft/util/*.java
```

The client code is available at this location (http://www.redbrick.dcu.ie/hego/project/gui_client1.tar.gz). Download this code into the root directory where weirdX is installed. Unzip and untar the file with the following command;

```
tar zxvf gui_client1.tar.gz
```

The java code is stored in the directory *com/jcraft/weirdx*. To compile this code type;

```
javac com/jcraft/weirdx/*.java
```

An RMI stub file then needs to be generated for the Spy class. This can be generated with the following command;

```
rmic com.jcraft.weirdx.Spy
```

A sample java.policy file is included with the client code. This does not need to be altered, unless you want to restrict permissions further.

Listing 4: java.policy

```
grant {  
    permission java.net.SocketPermission "*:1024-65535", "connect,accept";  
    permission java.net.SocketPermission "*:80", "connect";  
};
```

Finally, a sample client file that starts the client code is included called "client.bat". This should again be modified to reflect the system.

Listing 5: compile.bat

```
java -Djava.security.policy=java.policy com/jcraft/weirdx/StartClient
```

2.3 Using the X-Server

The client application consists of two panels. The first is the X-Server panel. An application can be run on the server, and the graphical output is redirected to the java application. This is very useful when one wants to run graphical programs from a machine that has no graphical output, ie., one can only log into it using a console. The following is an example of how to use the X-Server.

To access the server through a terminal, it is not necessary to log onto the server. If the server starts a terminal, then it will appear in the X-Server tab of the client application. To do this, the server must know where the client application is. This can be done by pointing the system *DISPLAY* variable to the ip address of the client machine, as in the following example;

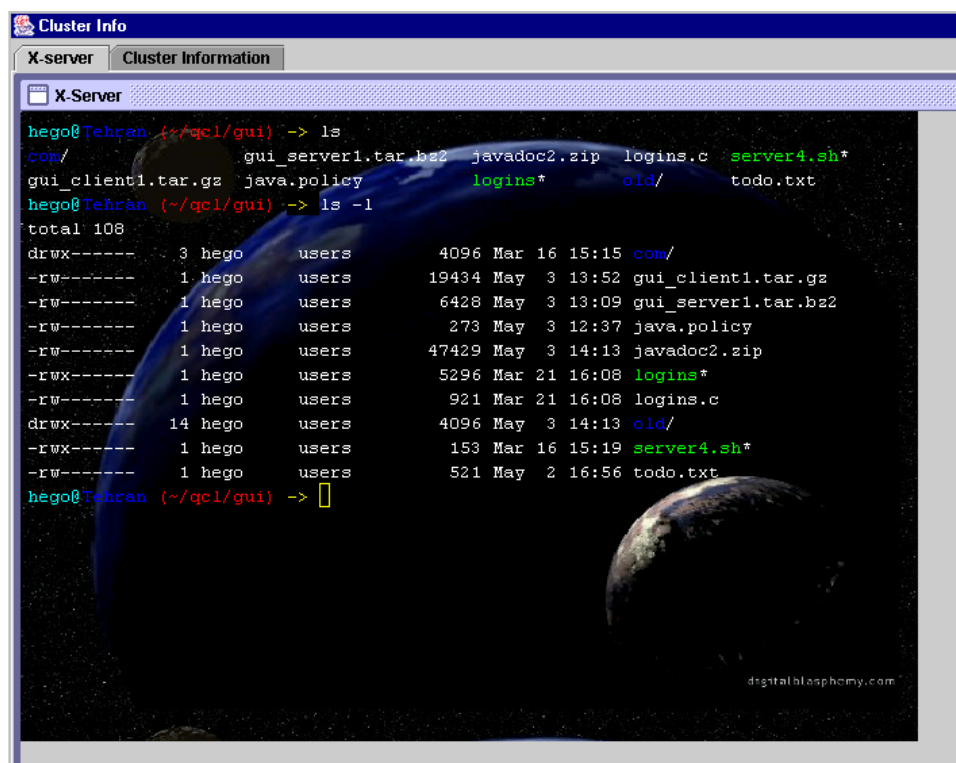
```
export DISPLAY=136.206.18.54:2.0
```

The number after the colon is the port number that the X-Server is running on.

To run a remote terminal on the client X-Server, the following should be typed into the server;

Eterm &

This produces the following picture on the X-Server panel of the client application;



```
Cluster Info
X-server Cluster Information
X-Server
hego@Tehran: (/qcl/gui) -> ls
com/          gui_server1.tar.bz2  javadoc2.zip  logins.c  server4.sh*
gui_client1.tar.gz  java.policy        logins*       old/      todo.txt
hego@Tehran: (/qcl/gui) -> ls -l
total 108
drwx-----  3 hego    users      4096 Mar 16 15:15 com/
-rw-----  1 hego    users     19434 May  3 13:52 gui_client1.tar.gz
-rw-----  1 hego    users      6428 May  3 13:09 gui_server1.tar.bz2
-rw-----  1 hego    users       273 May  3 12:37 java.policy
-rw-----  1 hego    users     47429 May  3 14:13 javadoc2.zip
-rwx-----  1 hego    users      5296 Mar 21 16:08 logins*
-rw-----  1 hego    users       921 Mar 21 16:08 logins.c
drwx----- 14 hego    users     4096 May  3 14:13 old/
-rwx-----  1 hego    users      153 Mar 16 15:19 server4.sh*
-rw-----  1 hego    users       521 May  2 16:56 todo.txt
hego@Tehran: (/qcl/gui) -> 
```

One can type in unix commands, such as the *ls* command shown on screen, and the response to the command will appear on the screen. A more graphical example of the X-Server, is if the *wmaker* window manager is run from the server. This results in the following screenshot;



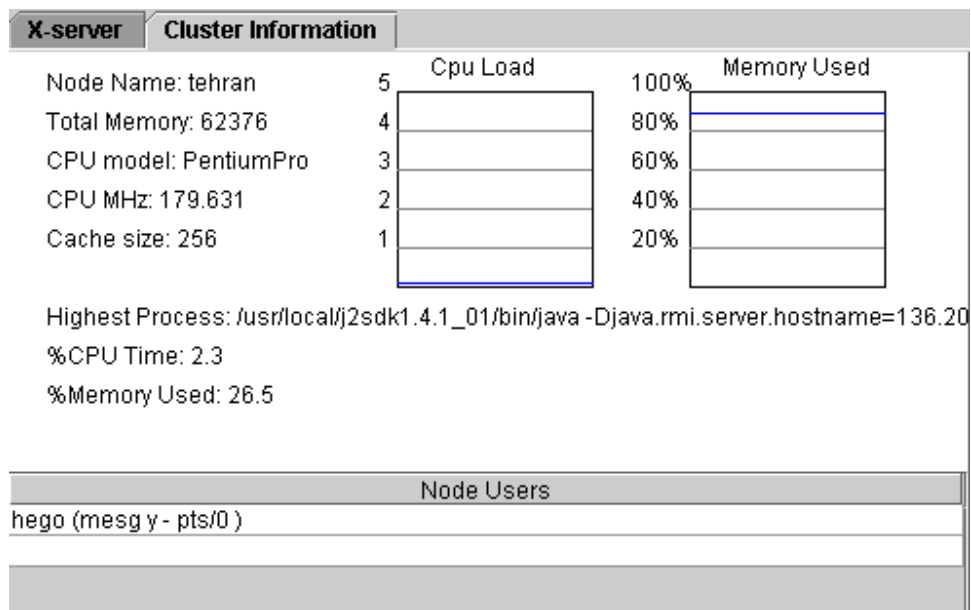
The X-Server is used in this project to allow the displaying of the Bloch Sphere graphical application.

2.4 The cluster information panel

The cluster information panel contains information about the cluster in general, and the nodes that comprise the cluster. The right half of the screen contains a table listing the nodes of the cluster, and various information about each node. This information consists of the node's status, ie. whether it is up or down, the node's one minute load, which is an indicator of how busy it is, the number of users logged onto each node, and the uptime of the node, which is the time from when it was last rebooted.

Node Name	Node Status	Load	Users	Uptime
tehran	up	0.12	2	up 3 days
tortilla	up	0.00	0	up 3 days
tambourine	up	0.07	0	up 3 days
doc	up	0.00	0	up 3 days
poprad	down			
sarv	up	0.01	0	up 3 days
eddie	up	0.07	0	up 3 days
akito	up	0.00	0	up 3 days
yoko	up	0.00	0	up 3 days
luce	up	0.00	0	up 3 days
darkschnieder	up	0.00	0	up 3 days
ruri	up	0.00	0	up 3 days
yurika	up	0.00	0	up 3 days
gendo	up	0.01	0	up 3 days
asuka	up	0.00	0	up 3 days
midori	up	0.00	0	up 3 days
hyuga	up	0.00	0	up 3 days
kebidabu	up	0.00	0	up 3 days
peanut	down			
teatime	up	0.00	0	up 3 days
loser	up	0.00	0	up 3 days
herschel	up	0.00	0	up 3 days
odie	up	0.00	0	up 3 days

The left hand panel is composed of three parts. The first is information about the currently selected node. You can change the currently selected node by clicking on the appropriate row of the table in the right hand panel. This information comprises the node name, the total amount of memory it has, the CPU model name, speed in megahertz and cache size. There is also two moving graphs which illustrate the load of the CPU over time, as well as the amount of memory that is used, as a proportion of the total memory. Finally, the highest process on the currently selected node is displayed, as well as the percentage of CPU time and percentage of total memory that process is using. The middle panel shows the names of users that are logged onto the root node and whether they have messages turned on or off.



The bottom panel on the left hand side shows general information about the cluster. This is comprised of the number of nodes of the cluster, the ip address of the head node, the operating system being used on the cluster, and the operating system version being used, ie. the kernel release number.

General cluster information:	
No. of nodes:	23
IP address:	136.206.19.200
Operating System:	Linux
OS release:	2.2.19pre17-compact

3 Using the parallel routines of the Quantum Computing Language

Five different parallel routines were written for this project and integrated into the QCL ¹.

3.1 Installing the MPI routines in QCL

To patch a previous QCL distribution, see the next section. This section details how to install a complete QCL application with MPI routines already built in. First of all, download the code from this location (http://www.redbrick.dcu.ie/hego/project/mpi_qcl_final_1.tar.bz2). Unzip and untar this file in the following way;

```
tar jxvf mpi_qcl_final_1.tar.bz2
```

Change directory into the directory that is created. The first thing to do is to make the dependencies. Type;

```
make dep
```

The final step is to compile it. Enter;

```
make
```

3.2 Installing the MPI patchfile to the QCL

To patch the MPI routines to the QCL code, download the most up-to-date version of the QCL(QCL-0.5.0) code to a directory. Download the mpi patch from the following location (http://www.redbrick.dcu.ie/hego/project/mpi_qcl.patch). First of all, untar and unzip the QCL code with the following command;

```
tar zxvf qcl-0.5.0.tgz
```

Change directory into the qcl-0.5.0 directory. Then apply the patch with this command;

¹See the technical manual

patch -p1 <(location of patch file)

To compile the newly patched source, issue the following commands;

```
make dep  
make
```

3.3 Using the MPI routines in the QCL

The qcl must be run with the *mpirun* command, passing it the number of processes to use with the *-np* command-line switch.

```
mpirun -np <nodes> -machinefile <nodes file> ./qcl
```

Five parallel routines are provided with the MPI extension to the Quantum Computing Language. The functions are called ParMatrix1-5 and all of the functions take a quantum register and a matrix as parameters. The following is an example of how to apply a parallel matrix decomposition to a trivially small qubit;

```
qureg a [2];  
Mix(a);  
Rot(0.6435, a);  
complex matrix m[4];  
m[0,0]=1  
m[1,1]=1  
m[2,2]=1  
m[3,3]=-1  
ParMatrix3(m,a);
```

4 A tutor on the Quantum Computing Language

The Quantum Computing Language is a programming language designed to approach quantum computing programming, using the syntax of a procedural language like "C". Quantum Computing is still seen as being in the domain of Theoretical Physics, rather than Computer Science, due to the abundance of different notation/formalisms. The Quantum Computing Language attempts to unify the

Quantum Computing concepts into a single coherent language. The Quantum Computing Language provides a base-set of operators, yet is able to simulate every known quantum algorithm that can be run on a Quantum Computer. The QCL was written by Bernhard Ömer from the Technical University of Vienna, and is released under the GPL(General Public License). It runs under a variety of UNIX platforms.

4.1 Classical components of the QCL

The Quantum Computing Language contains a number of classical programming components. The term *classical* refers to non-quantum components. These components are very similar to programming components found in a language like "C". The QCL has 5 data types; *int*, *string*, *complex*, *boolean*, *real*, all of which are fairly self-explanatory. The QCL also implements a full range of logic and arithmetic operators, as well as a large amount of mathematical functions, eg. *sin()*, *cos()* etc. The following example shows conditional branching in the QCL;

```
if e == -1 {  
    print "Unable to find a relatively prime number";  
    exit;  
}
```

The QCL provides three types of loops. The *for* loop is similar to a *for* loop in C. It also provides two conditional loops, a *while* loop and an *until* loop. The following piece of code is an example of a QCL *for* loop;

```
complex matrix m[1024];  
int i = 0;  
for i=0 to 1023 {  
    m[i,i] = 1;  
}
```

The QCL also provides a means of grouping statements into blocks and functions. The following is an example of calling a function from another piece of code;

```
int i = 0;  
int n = afunction(i);
```

```
...  
int afunction(int n) {  
    ....  
}
```

4.2 Quantum components of the QCL

The QCL is started with a set number of qubits in the system. The default number of qubits is 32. Qubits are manipulated by declaring quantum registers of an arbitrary number of qubits. An operator can then be applied to the quantum register.

```
qureg a[5];    // Initialise a quantum register of size 5 qubits  
Mix(a);        // Superpose all the qubits in the quantum register
```

The QCL provides many operators for quantum registers, a few examples are provided here;

- Rot(real theta,qureg q) - Rotate a single qubit through an angle
- Mix(qureg q) - Apply a Hadamard gate to the quantum register argument
- CPhase(real phi,qureg q) - The conditional phase operator
- Swap(qureg a,qureg b) - Exchange the qubits of two registers
- Not(qureg q) - Invert a qubit

The QCL provides two non-unitary operators as well. These operators are allowed, because they destructively interfere with the machine state instead of modifying it. The first operator, *measure*, is the equivalent of *observing* a qubit. It forces the qubit in question to decompose into one of its base states. The second operator is *reset*. This serves to reinitialise every qubit in the system.